Consider positive integers n and K. Does n have a factor (other than 1 or n) of value at most K? Let's write an algorithm for for this.

```
Factor (n, K)
    For i = 2 to min(K, n-1)
    |   If i | n
    |   |   Return true
    Return false
```

How long does this algorithm take to finish?

$O(f \min(K, n-1))$, where f is however long the algorithm takes to test divisibility. More importantly, what is the input size? It's $m = \log(\min(n-1, K))$, so if we express the runtime in terms of the input size m, we get $\Omega(2^m)$ (in the worst case). That means this is an exponential time algorithm. But giving an exponential time algorithm for a language is usually a good sign that it's in NP.

Formally, we define

$$\text{IntFact} = \{\langle n, K\rangle \mid n \text{ has a factor at most } K \text{ and not } 1 \text{ or } n\}.$$

We can easily give a verifier V for this language.

V = "On input $\langle\langle n,K\rangle, c\rangle$,
1) Check that c is an integer
2) Check that $1 < c < \min(k, n-1)$
3) Check that $c \mid n$
4) If any check fails, reject
5) Accept"

If n has a nontrivial factor $k_0 \leq k$, then $c = k_0$ will make V accept.
If n has no such factor, V never accepts, or rather it always rejects.

Now let's think about $\overline{\text{IntFact}}$. We can write an algorithm for this easily.

```
NoFactor(w)
  If w ≠ ⟨n,k⟩
    Return true
  Return !Factor(n,k)
```

This is still an exponential time algorithm, of course. So we should be able to write a verifier or a polytime NTM decider for the language.

$\overline{V}$ = "On input $\langle w, c\rangle$,
1) If $w \neq \langle n,k\rangle$, accept
2) Run $V(\langle\langle n,k\rangle, c\rangle)$
3) If V accepts, reject; if V rejects, accept"

Easy, right?
Nope. This doesn't work. V will tell us if c is a factor of n at most k and not 1 or n. It doesn't tell us if n has any other factors, and we need to know that n has <u>no</u> factors.

Okay, let's try a different approach.

N = "On input $\langle n,k \rangle$

1) Nondeterministically generate a number K_0 between 2 and $\min(n-1,K)$ (inclusive)
2) If $K_0 | n$, accept
3) Reject"

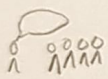
We have $L(N) = IntFact$, and it runs in poly time and always halts, so we again have $IntFact \in NP$.

Now let's use N to write a polytime nondeterministic decider for $\overline{IntFact}$.

$\overline{N}$ = "On input w,

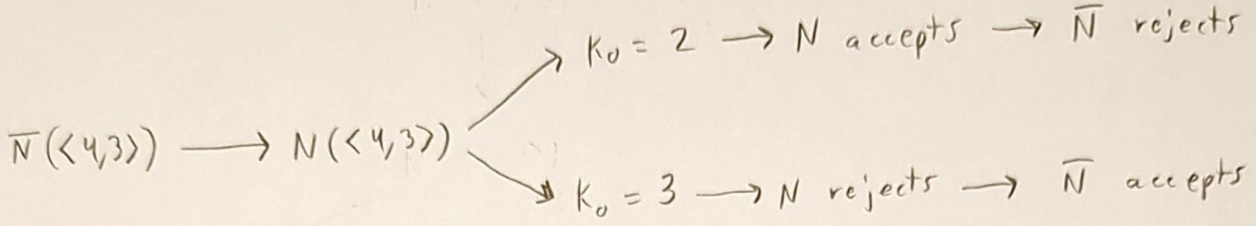
1) If $w \neq \langle n,k \rangle$, accept
2) Run $N(\langle n,k \rangle)$
3) If N accepts, reject; if N rejects, accept"

Okay, now this works, right? N will tell us if n has any nontrivial factors at most K, right?



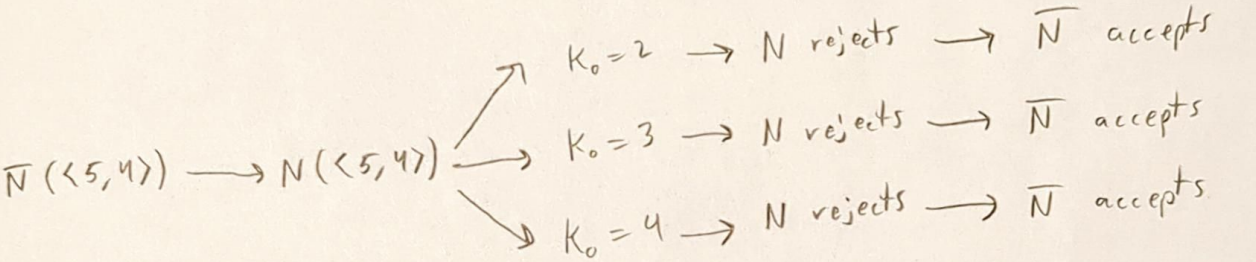
Well, no. Why not?

Think about what happens when you simulate a NTM. Essentially, you just inject its code into your machine. It does its thing once and then you move on. It makes you nondeterministic, though, if you weren't already, so now you explore every possible computation route. Let's look at what this does to $\overline{N}$ on an example input of $n = 4$ and $k = 3$.



Since there is a branch of computation that results in $\overline{N}$ accepting, $\overline{N}$ accepts $\langle 4,3\rangle$. This is a problem!

On the other hand, consider $n = 5$, $k = 4$.



This illustrates the trouble of flipping the output of an NTM. An NTM accepts if $\exists$ a computational branch that accepts. Further, it rejects if $\forall$ computational branches, it rejects.

If we flip the output, we get a new NTM that, intuitively, wants to reject if $\exists$ a computational branch in the original NTM which accepts and to accept otherwise. Instead, we accept if $\exists$ a computational branch in the original NTM that rejects and we reject if $\forall$ computational branches, the original NTM accepts.

We swapped our quantifiers! That doesn't work with the definition of nondeterminism, so N is (usually) not helpful for writing $\overline{N}$. This leads us to a big open problem in computer science.

We define the set co-NP similarly to co-RE (or indeed any general class of language $\mathcal{C}$),

$$\text{co-NP} = \{L \mid \overline{L} \in \text{NP}\}.$$

The big question, just as we don't know if P = NP, is if NP = co-NP.

As an aside IntFactor $\in$ co-NP as well. Showing this is somewhat tricky and involves number/group theory. Regardless, we know that NP $\cap$ co-NP $\neq \emptyset$.